Copy No. 6

DEFENCE DÉFENSE

SONAR IMAGE PROCESSING SYSTEM II UPGRADE PHASE 1: SELECTION OF COMPUTING PLATFORM

Barrodale Computing Services Ltd.

DEFENCE RESEARCH ESTABLISHMENT ATLANTIC

Contractor Report

DREA CR 1999-039

May 1999

National Defence Défense nationale

Canada

National Defence
Research and Development Branch

Défense nationale
Bureau de recherche et développement

DREA CR 1999/039

SONAR IMAGE PROCESSING SYSTEM II UPGRADE PHASE 1: SELECTION OF COMPUTING PLATFORM

Barrodale Computing Services Ltd.
P.O. Box 3075, Victoria, BC
V8W 3W2

Scientific Authority [illegible]
J. Fawcett

May 1998

W7707-8-6103/001/HAL
Contract Number

CONTRACTOR REPORT

Prepared for

Defence Research Establishment Atlantic

Centre de Recherches pour la Défense Atlantique

DEFENSIO PER SCIENTIAM

Canada

Abstract

Barrodale Computing Services, under contract to DREA/EDRD, had previously implemented sonar image processing algorithms on a DEC/Unix platform utilizing the ILLUSTRA database system and ENVI/IDL software. Since the end of those contracts, there have been changes to the database and image processing packages as well as a significant increase in computing power available in relatively inexpensive PC-platforms and workstations. The goal of this contract was the porting of the previously developed software to new, more powerful computers utilizing the current versions of the database and image processing software. The contract consisted of two phases: (1) the analysis of the best hardware-software configuration for the system (2) the purchase of the suggested hardware/software and the implementation of the sonar image processing system. The first part of the report – Phase I describes the various computer options for the system in terms of cost and performance. The second part of the report – Phase II- describes the implementation of the sonar imaging software on the new system, includes a user's manual, and has suggestions for future work.

Résumé

La compagnie Barrodale Computing Services, dans le cadre d'un marché conclu avec le CRDA/DRDE, avait précédemment mis en œuvre des algorithmes de traitement des images sonar sur une plateforme DEC/Unix au moyen du système de base de données ILLUSTRA et du logiciel ENVI/IDL. Depuis la fin de ces marchés, des changements ont été apportés aux ensembles de base de données et de traitement des images alors que la puissance de calcul des plateformes sur micro-ordinateurs et stations de travail relativement peu dispendieuses a augmenté considérablement. L'objectif de ce marché était de porter le logiciel développé précédemment à de nouveaux ordinateurs plus puissants utilisant la version courante du logiciel de base de données et de traitement des images. Le marché portait sur deux phases : 1) analyse de la meilleure configuration matérielle-logicielle pour le système, 2) achat du logiciel et du matériel recommandés et mise en œuvre du système de traitement des images. La première partie du rapport, portant sur la phase I, décrit les divers choix d'ordinateurs possibles sur les plans des coûts et des performances. La deuxième phase du rapport, portant sur la phase II, décrit la mise en œuvre du logiciel d'imagerie sonar sur le nouveau système, comprend un manuel de l'utilisateur et comporte des suggestions relatives aux travaux ultérieurs.

Table Of Contents

Sonar Image Processing System II Upgrade

Phase 1: Selection of Computing Platform

1. Introduction

Previous Department of National Defence (DND) contracts with Barrodale Computing Services Ltd. (BCS) have led to the partial development of a sidescan sonar processing/minehunting system (SIPS-II - Sidescan Image Processing System, hereafter referred to simply as "SIPS") based upon ENVI-IDL signal/image processing and the Illustra database software. The system was developed to provide a modern, state-of-the-art research tool for the investigation of various issues related to the processing and analysis of Remote Minehunting System (RMS) data. This system was implemented during 1995/96 on a DEC-Alpha workstation with the Unix operating system. Since the original contracts concluded, there have been changes to the ENVI/IDL and Illustra software (for example, Illustra has since been bought by Informix Software, Inc.). It is also thought that it would be advantageous to use either a PC- or Sun-based system due to the higher-version level of Informix's database software for these platforms. For this reason, it has been decided to consider upgrading the current SIPS system. This plan would be implemented in two phases, the second phase being contingent on the results of the first.

1) In Phase I, a new hardware platform for SIPS would be recommended, and
2) In Phase II, the SIPS software would be ported to this new platform.

This report investigates the pros and cons for several possible computer configurations and makes recommendations to guide a decision on which platform to adopt for SIPS. Please note that all prices shown are in Canadian dollars, unless otherwise indicated.

2. Review Criteria

The SIPS-II Upgrade RFP suggested the following criteria:

(1) estimated costs of required software upgrades – costs of upgrades in the future,

(2) costs of required computer system and operating system
  – details on CPU speed, memory, and monitor requirements,

(3) best computer strategy in terms of future computational requirements. For example, a real-time implementation would require certain processing and display programs to be constantly running while other analysis programs were being performed by the system operator. Can the software development be flexible enough that additional processing and displays (perhaps, through additional CPUs) could be easily added in the future?

(4) best computer/operating/graphics system for any future real- or near real-time communications.

(5) best computer/software/graphics strategy to allow for easy porting of developed software system to other computer platforms.

3. Conversion Issues

3.1 Illustra - IDS/UDO Conversion

3.1.1 Identifier Naming

In Illustra, identifiers (e.g., table names and column names) were case-sensitive and not limited in length. In the current version of Informix (Informix Dynamic Server - Universal Data Option, or IDS/UDO), identifiers are limited in length to 18 characters[1] and are not case-sensitive. Case sensitivity will not be a conversion issue, since no two distinct tables in SIPS have names that differ only in their case. However, several SIPS database objects have identifiers longer than 18 characters and will have to be renamed. Renaming these database objects will of course result in numerous minor changes to C, IDL, and SQL source code.

3.1.2 SQL Syntax

Several Illustra data types and SQL language constructs are not supported, or are supported differently in Informix than they were in Illustra. Fortunately, none of the discontinued/unreplaced data types is used in SIPS. On the other hand, a significant amount of the current SQL code (which is in turn embedded in C and IDL source code) will need to be rewritten to conform to the modified syntax and new data type names.

[1] Version 9.2 of the Universal Data Option (due to be released in late 1999) will support identifiers up to 128 characters in length.

3.1.3 Large Object Support

Large objects are supported slightly differently in Informix than they were in Illustra, and some of the C and SQL code that deals with large objects will have to be modified. Informix supplies a "Large Object Locator Datablade"; this "bundled" datablade should help simplify the current source code.

3.1.4 Conversion of Illustra Data

The current SIPS database consists of 64 tables, many of which have composite data types and large object data types. Illustra provides server functions for copying large object data out into files. These server functions, in conjunction with simple SQL code, can be used to export the current database contents to the file system. From the file system, this data can then be loaded into a new IDS/UDO database.

3.2 IDS/UDO System Dependencies

3.2.1 IDS/UDO Features Supported on Unix But Not on NT

There are several IDS/UDO features that are available in the Unix version but not in the NT version. This situation is partly due to the difference between the Unix and NT process models, and the way Informix has chosen to implement database server processes using the underlying operating system (i.e., as Unix *processes* in Unix and as NT *threads* in NT). The features not supported in the NT version are:

- Processor affinity: the ability to bind a particular server process to a particular CPU (this is only relevant, of course, in multiprocessing systems).
- Process aging: the ability to prevent the operating system from reducing the priority of a server process over time.
- Multiple residency: the ability to run multiple Informix instances on the same machine. This is really only desirable in large installations with lots of development activity where one wants to isolate development activity from production use. In an NT environment, the same goal can be cost-effectively achieved by hosting the different instances on different machines.
- Use of raw disk space: in Unix, Informix can be configured to bypass the file system and read and write directly to the disk hardware. This facility has benefits, in terms of both performance and data integrity. The facility is not available under NT.

Fortunately, none of these IDS/UDO features that are available in the Unix version but not in the NT version are relevant to the SIPS software.

3.3 IDL/ENVI System Dependencies

3.3.1 Communication Between IDL and IDS/UDO

In the existing SIPS system, two different methods of communication are used to pass data between IDL and Illustra. In the first method, which will be referred to as the socket method, a process is spawned from IDL (using the IDL **SPAWN** command) and an IDL input/output (I/O) unit number is assigned to handle the flow of data between IDL and the spawned process. Hence, input to the spawned process from IDL and output from the spawned process to IDL are achieved by means of the assigned I/O unit. Once spawned, this process calls Illustra functions to open a connection to the database and it listens for SQL commands on its input stream. The spawned process uses Illustra functions to submit commands to the Illustra server and to retrieve the results which are output through the socket to IDL. The socket method can only handle Illustra base data types and composites of these data types; it cannot handle large objects.

The second method, referred to as the **CALL_EXTERNAL** method, is used to transfer data between IDL and Illustra large objects. This method, which is based on the IDL **CALL_EXTERNAL** facility, consists of a library of C functions compiled as shared objects.

The use of the IDL **SPAWN** mechanism is platform-dependent. While **SPAWN** is available in both NT and Unix versions, the ability to communicate via sockets with the spawned process is available only in Unix versions of IDL. Fortunately, the **CALL_EXTERNAL** facility is not platform-dependent, and this method can handle all IDS/UDO data types, not just large objects. Because the invocations of the socket method in the current SIPS system are limited to a small set of simple IDL routines, it should be a simple matter to change SIPS to use the **CALL_EXTERNAL** facility universally. While this conversion would only be necessary if SIPS were to be ported to NT, we feel it would be beneficial, both from the performance aspect and the maintainability aspect, to perform this conversion even if SIPS were to remain on a Unix platform.

3.4 External Program System Dependencies

3.4.1 ADA Programs

3.4.1.1 Demultiplexing Navigation MUX Data Module

This module includes 1200 lines of ADA source code. An initial examination of this code reveals that it is fairly standard and doesn't contain any DEC or Unix platform dependencies, so compiling and executing this code on an NT or Sun platform should pose no problems. Commercial ADA compilers are available on both platforms, and the GNU project (http://www.gnu.ai.mit.edu/gnu/gnu-history.html) has produced a free ADA compiler that runs under NT and Sun. (see http://louisa.levels.unisa.edu.au/ada/gnat/compilers.htm)

3.4.1.2 Navigation Estimation Module

This module includes 750 lines of ADA source code. This code appears to be fairly generic, although there is a DEC-specific I/O ADA package that is invoked. Some effort will be required to port this short package to the platform selected.

3.4.2 Awk Scripts

In the current SIPS system, two short, simple scripts (approximately 200 lines total), written in "awk," are used to convert Illustra table definitions to IDL data structure definitions. These scripts serve as development tools and are not really part of the SIPS production system. Nonetheless, they should be kept to assist in maintenance activities in the future. If a Unix platform is chosen, they can remain as awk scripts; if an NT platform is chosen then they could easily be translated manually to C.

3.4.3 Fortran Code

3.4.3.1 Least Squares Spline Library

This library consists of a 195-line Fortran IV subroutine. This subroutine should easily be convertible to C. BCS has experienced favorable results in using the public domain translator **f2c** (see http://www.netlib.org/f2c/) on subroutines such as this one. It would be our recommendation (in the interest of cost and maintainability) to convert this Fortran subroutine to C using f2c.

3.4.3.2 UTM Library

The UTM library consists of three Fortran subroutines, totaling 563 lines. Conversion of these subroutines to C using the f2c program should be straightforward.

3.4.3.3 Source Sonar Image Normalization Module

This module uses subroutines contained in a set of four large files of Fortran source code (total of 1759 lines of code). While all of this code has been linked into a shared object library, many of the procedures in these files do not appear to get called, and some of the procedures call other procedures for which source code is absent. We would suggest that in Phase II, the lines of code not actually being used by SIPS be removed; the remaining code should then be converted to C using f2c.

4. Potential Configurations

4.1 IDS/UDO-Supported Configurations

Currently, IDS/UDO is available in an NT/Intel version and a Solaris/Sun version. A Solaris/Intel version will be available early next year.

4.2 IDL/ENVI-Supported Configurations

IDL and ENVI are available in the following Operating System / Hardware platforms relevant to this project: NT/Intel, Solaris/Intel, Linux/Intel, and Solaris/Sun.

4.3 Summary

As is the case with most other DBMS's, IDS/UDO operates in a client-server fashion. The database server runs on one machine (the server machine), while one or more SQL clients run on one or more (possibly) other machines (the client machines). There is nothing inherent in IDS/UDO client programs (such as SIPS) requiring the clients to run on the same machine as the database server, and there is nothing to prevent the clients from running on the same machine. Hence, given the availability of the software on which SIPS is built, there are actually eight possible configurations:

IDL/ENVI	IDS/UDO
NT/Intel	NT/Intel
Solaris/Intel	NT/Intel
Linux/Intel	NT/Intel
Solaris/Sun	NT/Intel
NT/Intel	Solaris/Sun
Solaris/Intel	Solaris/Sun
Linux/Intel	Solaris/Sun
Solaris/Sun	Solaris/Sun

Given that there will only be one SIPS client at present, a multi-machine platform is not really warranted. However, if it is expected that SIPS will support multiple clients in the future, then there is nothing to prevent the database server from being moved to a new machine at that time. Such a change would require no additional porting effort (just the cost of the new hardware).

In summary, there are two basic configurations warranting further investigation:

i) NT/Intel
ii) Solaris/Sun

5. Shortlisted Configurations

5.1 Solaris/Sun Configurations

The following table illustrates a range of Sun platforms that would be suitable for SIPS. The Ultra 5 and Ultra 10 are entry-level, uniprocessor workstations. The standard configuration for the Ultra 5 comes with a relatively slow (4500 RPM EIDE) disk instead of the faster SCSI devices. To facilitate comparison between the Sun configurations and the Dell configurations in the next section, we have upgraded the Ultra 5's configuration to include a SCSI disk instead of the standard one.

The Ultra 30 and Ultra 60 are higher performance workstations, specifically configured for design, analysis, and visualization. The standard configurations include SCSI disk drives, and a second CPU can be added to the Ultra 60 for approximately $10,000.

The ENTERPRISE 250 is a server machine, intended to be used as the server in a client-server environment (e.g., as a database server, a web server, or a LAN file server). In the following chart, the configuration includes 2 CPU's.

All CPU's shown are 64-bit SPARC Version 9 UltraSPARC II CPU's. The prices shown are approximate and do not include taxes.

Name	CPU Speed	RAM (expand)	Disk	Disk Type	Disk Speed	Graphics Accel?	Monitor Size	Graphics Res.	Tape Drive	Approx. Price
Ultra 5	333 Mhz	128 (512)	9.1 GB	SCSI	7200 RPM	Yes	19"	1280 X 1024	4mm 12 GB	$10,000
Ultra 10	333 Mhz	128 (1024)	9.1 GB	SCSI	7200 RPM	Yes	19"	1280 X 1024	4mm 12 GB	$11,000
Ultra 30	250 Mhz	128 (2048)	9.1 GB	SCSI	7200 RPM	Yes	21"	1280 X 1024	4mm 12 GB	$21,000
Ultra 60	300 Mhz	128 (2048)	9.1 GB	SCSI	7200 RPM	Yes	21"	1280 X 1024	4mm 12 GB	$26,000
ENTERPRISE 250	2 X 250 Mhz	128 (2048)	9.1 GB	SCSI	7200 RPM	No	19"	1280 X 1024	4mm 12 GB	$23,000

Further details of these configurations can be found at the following web sites:

Ultra 5	http://nmso.sun.ca/cat1a.htm
Ultra 10	http://nmso.sun.ca/cat1b.htm
Ultra 30	http://nmso.sun.ca/cat3b.htm
Ultra 60	http://nmso.sun.ca/cat4b.htm
ENTERPRISE 250	http://nmso.sun.ca/cat14a.htm

5.2 NT/Intel Configurations

The following NT configurations are available from Dell Canada. All platforms include the Pentium II processor, except the Precision 610 which includes the more powerful Pentium II Xeon processor. The Optiplex is Dell's line of business workstations. It is included in this list because it is the only workstation line that appears to be included in Dell's Federal Government Standing Offer. The Precision line is Dell's line of high performance graphics workstations; these machines are roughly comparable to Sun's Ultra line. The PowerEdge computers are aimed at server applications. These machines are roughly comparable to the Sun Enterprise line.

Prices shown are approximate and do not include taxes. The Optiplex and PowerEdge 6300 computers are available under the Federal Government Master Standing Offer (NMSO).

Name	CPU Speed	RAM (expand)	Disk	Disk Type	Disk Speed	Graphics Accel?	Monitor Size	Graphics Res.	Tape Drive	Approx. Price
Optiplex Gx1M 6450 (NMSO)	450 Mhz	128	7.5 GB	EIDE	7200 RPM	Yes	17"	1280 X 1024	4 GB	$3585
Precision 610	2 X 450 Mhz	256 (1024)	9 GB	SCSI	7200 RPM	Yes	21"	1280 X 1024	4mm 12GB	$12,000
PowerEdge 2200	2 X 333 Mhz	128 (512)	9 GB	SCSI	7200 RPM	No	21"	1280 X 1024	4mm 12 GB	$8,000
PowerEdge 2300	2 X 450 Mhz	128 (1024)	2X4.5 GB	SCSI	7200 RPM	No	17"	1280 X 1024	4mm 12 GB	$10,000
PowerEdge 6300 (NMSO)	4 X 400 Mhz	256	9.1 GB	SCSI	7200 RPM	No	21"	1280 X 1024	4mm 12 GB	$18,000

Further details of these configurations can be found at the following web sites:

Optiplex Gx1M 6450	http://www.dell.ca/products/desktops/optiplex/gx1/gx1_specs.htm
Precision 610	http://www.dell.ca/products/workstations/610/610_specs.htm
PowerEdge 2200	http://www.dell.ca/products/servers/poweredge/2200/2200_specs.htm
PowerEdge 2300	http://www.dell.ca/products/servers/poweredge/2300/2300_specs.htm
PowerEdge 6300	http://www.dell.ca/products/servers/poweredge/6300/6300_specs.htm

6. Analysis of Shortlisted Configurations

6.1 Software Cost

Product	Sun	NT
IDS/UDO[2]	$3,657.50 US	$3,657.50 US
IDL/ENVI[3]	$1,750 US	$894 US
Gnat ADA	free	free
f2c	free	free
C	free (GNU C)	$600 US (Microsoft Visual C++ Professional Edition)

6.2 Hardware Cost

The hardware costs of the shortlisted configurations are shown in the charts in Section 5. The Intel-based computers (Section 5.2) tend to be less expensive than the Sun computers (Section 5.1). Price/performance is considered in the next section.

6.3 Performance

6.3.1 Server Performance

The Enterprise and the PowerEdge computers are the high server-performance offerings from Sun and Dell, respectively. Both machines are popular as web servers, file servers, and database servers. The relative strengths of these two lines, however, are unclear. (See, for example, http://marwww.marand.si/press/14jan98.html, where the Enterprise 450 beats out Intel, and http://www.mindcraft.com/whitepapers/nts4sol26exec.html, where Intel beats out the Enterprise 450 in file and web server performance and price-performance.)

[2] These prices include the cost of software maintenance for the period ending December 10, 1999. Software maintenance beyond that point is $1732.50 US per year (regardless of platform). The existing one-user Illustra license will be replaced by a five-user IDS/UDO license free of cost (except for these maintenance costs).

[3] These prices include the cost of software maintenance for the period ending December 31, 1999. Software maintenance beyond that point is $875 US per year for Sun and $447 US per year for NT. Technical support costs a further $400 US per year.

6.3.2 CPU/Graphics Performance

The server machines (Enterprise and PowerEdge) tend to have poorer CPU and graphics performance than workstation machines (Ultra and Optiplex/Precision), since their focus is on serving large numbers of clients and maximizing throughput.

Comparing the Optiplex and the Precision Dell lines, the Precision Dell line has superior graphics and CPU performance, as it is aimed at the scientific, visualization/design/CAD market.

According to http://www.intel.com/businesscomputing/wrkstn/mid_range/p2xp_risc.htm, the Pentium II Xeon processor (used in Dell's Precision line) performs very well, relative to the Sun Ultra 60 (and hence the less powerful Ultra's), in benchmarks aimed at assessing performance on CAD, graphics, and visualization applications.

6.4 Flexibility

6.4.1 Adding Multiple Displays

The Sun Ultra 60 system supports use of dual monitors, through the addition of a second graphics card. The operating system can be configured to manage these multiple displays as a single large virtual desktop. Similarly, the Dell Precision 610, with the proper device driver and graphics cards, can support up to four monitors simultaneously.

6.4.2 Adding Multiple CPU's

The server machines (Enterprise and PowerEdge) tend to support multiple CPU's, since they are intended to support multiple clients and offer high throughput. The workstation machines, on the other hand, tend to be uniprocessor systems. The exception is the Dell Precision 610, which can support a second CPU.

Platform	Maximum Number of CPU's
Sun Ultra 5	1
Sun Ultra 10	1
Sun Ultra 30	1
Sun Ultra 60	2
Sun ENTERPRISE 250	2
Dell Optiplex Gx1M 6450	1
Dell Precision 610	2
Dell PowerEdge 2200	2
Dell PowerEdge 2300	2
Dell PowerEdge 6300	4

6.5 Real-time Communication

Real-time applications require a predictable, bounded latency between when an activity (e.g., data capture) is signaled to run (e.g., because data has just become available), and when the activity actually begins running on the processor. Both the Sun and NT operating systems support real-time to a degree, simply by virtue of the fact that they are pre-emptive, multiprocessing systems. The operating systems differ, however, in how well they can respond to events and ensure that adequate resources are made available to the processes handling the events.

6.5.1 Sun

According to Real Time Magazine's online edition (http://www.realtime-info.be/encyc/market/rtos/rtos/rtos75.htm),

> "The Solaris 2.x kernel is a pre-emptive multithreaded kernel, entirely different from the SunOS 4.x kernels. The new OS supports a real-time priority scheduling class above the normal Unix scheduling class. They quote a "guaranteed" worst-case response time (event-to-actual-process running) of under 1ms on a Sparc. The newer machines (Sparc 10) should do much better, but the hard 1ms response time is good enough to qualify it as a real-time OS."

6.5.2 NT

The NT operating system is inferior to Sun's Solaris operating system in at least two ways as far as support for real-time is concerned:

1) it has fewer priority levels for processes and threads (5 instead of 256), so it is harder for the operating system to distinguish between real-time and non real-time requests;
2) because there is no priority inheritance mechanism, the operating system can experience the problem of priority inversion (where a low priority process holds a resource that a higher priority process requires).

See Real Time Magazine's online addition for more information. (http://www.realtime-info.be/encyc/techno/publi/faq/rtfaq.htm#Windows NT)

6.6 Future Porting Flexibility

The following changes recommended in this report will make it easier to port SIPS to a different platform, regardless of the platform chosen now:

- convert Fortran code to C using f2c;
- replace use of the IDL **SPAWN** command with **CALL_EXTERNAL**;

- replace awk scripts with C programs.

If a Unix platform is chosen for Phase II, then work in the future to port SIPS to a different Unix platform would be minimal. If NT is chosen in Phase II and it turns out to be necessary to port SIPS to a Unix platform in the future, then more work will be required. However, because of the changes recommended above, the amount of work required would be much less than what would be undertaken in Phase II.

7. Recommendations

7.1 Choosing between NT and Sun

Our analysis has indicated that a port of SIPS to either an NT or a Solaris/Sun platform would be feasible. Price/performance considerations favor the Dell configurations. Another consideration in favor of NT/Intel platforms is the consistency of platforms. With an NT approach, output from SIPS will be available on the same desktop as other applications the research scientist user might be accessing. There would be no need to transfer results from one system for presentation on another.

The only factor favoring a Sun configuration is support for true real-time applications. This may or may not be an issue, depending on the intensity of the real-time applications, which is perhaps an unknown at this point.

7.2 Choosing between a Workstation and a Server

While SIPS would run on either a workstation or a server platform, we feel that a workstation should be chosen at this time, because of the intensity of computational and graphical requirements. If the SIPS database were to be shared by several clients, then the database server portion of SIPS (i.e., IDS/UDO) could be run on a separate server machine, as described in Section 4.3.

If a Sun platform is chosen, then the Ultra 10, 30, or 60 should be selected, depending on budget constraints. If an NT platform is chosen, then we would recommend purchasing the Dell Precision 610.

SONAR IMAGE PROCESSING SYSTEM II UPGRADE

PHASE 2: PORTING AND IMPLEMENTATION OF SYSTEM III

prepared for
Dr. John Fawcett
Defence Research Establishment Atlantic

by
Barrodale Computing Services Ltd.

P.O. Box 3075, Victoria, BC V8W 3W2

Contract Number W7707-8-6103/001/HAL

March 1999

TABLE OF CONTENTS

1 Background

The Sonar Image Processing System (SIPS) is a computer system used to process sonar images retrieved from field runs. The ultimate goal of SIPS is to produce an accurate image of the ocean floor scanned during a particular field run, and to analyze the resulting image. A *field run* consists of a sonar scan of the ocean floor by a vessel referred to as a *towfish* attached to a moving ship by a cable. The towfish has various sensors attached to it, including at least two (2) sonar sensors (or *channels*).

A digital data acquisition system on the ship is used to store sensory information returned by the various sensors of the towfish via the cable. Sonar imagery data is stored as synchronous data in a disk file referred to as a *raw sonar data file* (see [[3]] and [[6]] for details of the format of this file). This raw data, when viewed, will always look like a straight path. However, various factors make this image an inaccurate representation of the actual location of each pixel in the image. For example, the ship moves in different directions and also moves up and down on the water; the towfish also moves and has a roll, pitch and yaw; the ocean floor is not flat, nor is it parallel to the surface of the water.

These factors and others need to be considered when processing the raw sonar imagery data to produce a geographically correct digital representation of the ocean floor actually scanned. Typically these factors are determined by data returned by other sensors on the towfish. Such data is referred to as *ancillary data*. Unlike the raw sonar data, ancillary data is obtained asynchronously. That is, each sensor returns data at different rates, and the data returned by each sensor is of a different type and has a different format. All data from all sensors for a particular run is multiplexed and stored in a *mux file*. Each record of the mux file contains various header information such as the time index of the data, the data type and the data format, as well as the data itself. Note that in order to use any ancillary data, it has to first be synchronized. The **Navigation Estimation** module of SIPS performs, among other things, a synchronization on all ancillary data to produce a time series for each type of data.

The raw sonar data file and the mux file together form the *field data* for a particular run. In SIPS, C and Ada code is used to read data from these files and to do some of the processing of the sonar image. In particular, *demultiplexing* of the mux file is a complicated procedure and is achieved using an Ada program. However, the majority of the code, including the graphical user interface of SIPS is written in **IDL**. Algorithms used to process the data are written usually in IDL, but can be written in any language. SIPS uses an **Informix** database to store both intermediate and final output data (see [[3]], [[4]], [[6]] and [[7]] for details of the database design and implementation).

The reason for using a database, other than to keep all the data in one place, is to enforce some level of integrity on the data and to assist the user in following a logical sequence of processing tasks. Once sensory data has been associated with a raw sonar image in the database, the database will prevent the user from selecting algorithms that need input that do not yet exist. For example, geocoding an image requires towfish track estimation data, which in turn requires towfish roll, pitch and yaw data. Towfish roll, pitch and yaw data may not have been acquired during the run, but there may be other data that can be used to determine these parameters using, for example, the *dead reckoning* algorithm. The database will not allow the user to geocode an image then, unless and until the dead reckoning algorithm has been applied, the towfish roll, pitch and yaw determined, and the towfish track estimation performed. In fact, the dead reckoning algorithm may itself need other data that needs to be produced by yet another algorithm, and in such a case the database would not present the user with the choice of applying the dead reckoning algorithm until the required data has been produced.

Figure 1 shows the flow of data in SIPS. Note that the majority of the work is done by the graphical user interface and processing software (see [[6]] for more details). Once the raw sonar data has been input, and the ancillary data has been input, multiplexed and synchronized, the **Navigation Estimation** and **Towfish Track Estimation** modules are used to produce a georeferenced image. In a *georeferenced* image, the physical geographic location of each pixel in the image is known. The georeferenced image (or an appropriate section of it) is then geocoded to produce a geocoded image. In a *geocoded* image, not only is the location of each pixel known, but each pixel is in the correct position relative to the entire image, thereby showing an accurate representation of the area originally scanned in the field (see [[4]] for more information on the geocoding algorithm).

See [[6]] and [[7]] for a more detailed explanation of the overall design of the SIPS II system.

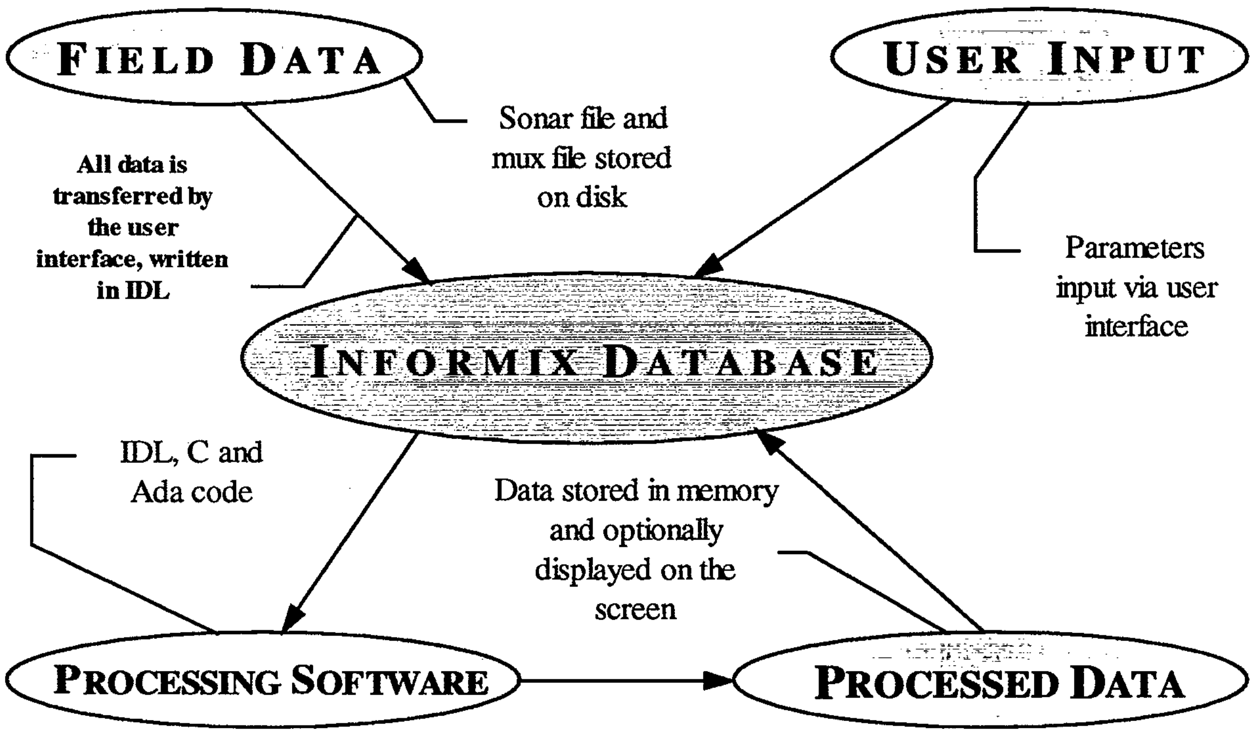
FIELD DATA
USER INPUT
Sonar file and mux file stored on disk
All data is transferred by the user interface, written in IDL
Parameters input via user interface
INFORMIX DATABASE
IDL, C and Ada code
Data stored in memory and optionally displayed on the screen
PROCESSING SOFTWARE
PROCESSED DATA

Figure 1: Flow of data in SIPS III

2 Using SIPS III

As mentioned in Section 1, the goal of SIPS is to produce an accurate image and to analyze it. Currently in SIPS III, the final image produced for any given run is a geocoded image. A typical session, then, in SIPS entails the following steps (in Windows NT):

1. Obtain the input data in the form of a sonar file (usually with a `.son` suffix) and a mux file (usually with a `.mux` suffix and the same name as the sonar file). For example, the files `run_052.son` and `run_052.mux` would form one set of input data.

2. Code any new processing algorithms. Refer to the `alg_dead_reckoning.pro` and `alg_dead_reckoning_apply.pro` files in the `src\sips\navestimate_module` sub-directory of the main sips directory for an example of how to do this. References to new algorithms should also be put into the database (see Section 3 and [[3]] for more details). *This step is optional.*

3. Log in as user `sips` and use the **SIPS** item in the **SIPS III** group of the **Programs** menu in the **Start** menu to start SIPS.

4. Using the **Control Panel**, specify which files to consider as choices for input data. Note that if the "Node names" value is set, all I/O will occur over the network and hence be considerably slower. If the file is on the local hard drive, it is recommended that this field be left blank.

5. Use the **Input** module to specify a *survey pass*. At the very least a name and a location (in latitude/longitude coordinates) has to be associated with the pass. Other data should also be input to fully identify the survey pass (which corresponds typically to a field run).

6. Use the **Normalization** module to calculate the *source altitude* and possibly to normalize the image itself. Before performing a normalization, a *compression algorithm* must be chosen. Also, a subset of the image (rather than the entire image) can be normalized. In addition, both a *gain* and *offset* function can be applied to each channel in the image. After performing the normalization, the normalized image and/or the source altitude must be stored in the database. Saving the normalized image automatically saves the source altitude. The raw sonar image, the subsampled image and the normalized image can all be displayed at this point. The **Display** part of the **Image Processing** module can also be used to view these images once they have been stored in the database.

7. Demultiplex the ancillary data into a set of time series using the **Demultiplexing** module. Each series can be plotted and manipulated to produce new time series if needed. The resulting time series must be saved to the database before continuing. One or more time series can be plotted, and two (2) time series can be plotted against each other if needed at this point.

8. Process the ancillary data using the **Navigation Estimation** module. The first step is to synchronize the time series since they were acquired asynchronously. This is done using the "`Process Time Series`" button. Then any relevant algorithms must be applied to the time series. Each algorithm takes as input some subset of the demultiplexed time series, and outputs new time series. Currently, the following algorithms exist in the system: **ship track fix, towfish heading fix, towfish position fix** and **dead reckoning**. In a typical session with the currently existing data, the algorithms should be applied in that order. Note that there are two (2) ways to apply an algorithm: using *only viable* algorithms or using *any* algorithm. The database is used by the user interface to determine which algorithms are viable for the current set of time series at any given time, and only those are presented to the user (unless the user chooses to apply *any* algorithm). Again, this ensures the integrity of the data and the process. At any point, any time series available (demultiplexed, synchronized or processed) can be plotted if needed.

9. Use the **Towfish Track Estimation** module to produce a georeferenced image. The **Display** part of the **Image Processing** module can be used now to view this image.

10. Use the **Geocoding** module to produce a geocoded image (see [[4]] for more information on how the geocoding is actually performed). Each channel in the sonar image must be geocoded separately. The outline of the sonar image (track) is plotted in a window. Use the mouse to specify a rectangular area to be geocoded, then select the `Geocode` button. The area can be a portion of the outline of the image, the entire image, or even an area larger than the image. Note that the larger the area to be geocoded, the longer it will take to perform the geocoding and the larger the resulting image will be. Once the geocoding has finished, the resulting image must be saved to the database. The geocoded image can be viewed directly at this point, or by using the **Display** part of the **Image Processing** module.

The **Image Processing** module can be used to display images, produce image histograms, produce image profiles, navigate through images, or to subsample images. It operates on normalized images, georeferenced images and geocoded images.

Note that the underlying database tables can be viewed and deleted from the user interface, but most database changes should be made using another tool such as **dbaccess** or **SQLEditor**. ***Any such change should be done with great caution. It is recommended that a full backup of the database be done before making any major change to it.***

The system has been set up to perform daily backups of the file system on to 4mm 12GB tapes. Also, icons have been set up in the **SIPS III** group of the **Programs** menu in the **Start** menu to start programs that perform backups and restores of the database. It is not currently possible to combine file system and database backups on the same tape.

If the database becomes completely corrupted, the database should be restored from tape. Also note that whenever the system is rebooted, the user must do the following:

1. Log in as user **administrator.**
2. Open the **Control Panel** item from the **Settings** menu in the **Start** menu.
3. Open the **Services** item in the **Control Panel.**
4. Find the two items that begin with the word "**INFORMIX**", and for each one select it and press the **Start** button.
5. Close the **Services** window.

More detailed information on using the system and the functions of the various modules of SIPS can be found in [[3]], [[4]] and in other sections of this document.

3 Database Design

The development of SIPS II involved designing a new database using **Illustra** as the database management system (DBMS). While SIPS III uses **Informix** as its DBMS, the high level design of the database is very similar to that of SIPS II.

The relationship between different data items and the related hierarchical structure is implemented by using sub-tables and super-tables. While in SIPS II uniqueness of primary keys was achieved using triggers, the same effect is achieved in SIPS III without the use of triggers (see Section 5.2).

Images, time series, tabular function values and other mathematical arrays were implemented in SIPS II using the Illustra *large object* data type and the *array* data type. In general, SIPS III also uses the Informix large object data type and array data type. While SIPS II used the **Poly** data type of the Illustra **2D spatial datablade** to represent the perimeter of images, SIPS III defines its own version of Poly since Informix does not have an equivalent to the Illustra 2D spatial datablade.

Note that the database is created and populated with default data using a series of SQL scripts contained in the **db** sub-directory of the main sips directory. These scripts create the database, create row types, casts, tables and views, and insert initial data into the appropriate tables. Views are used to facilitate performing complex SELECT statements from more than one table.

3.1 Terms and concepts

The following section gives brief descriptions of various terms and concepts used in the design of the database. More information can be found in [[3]].

Source sonar data file

A sonar file is essentially a binary file of fixed length records, where a single record contains the data from a single scan of all the *transducers* or channels during a field run.

Geometry

Much of the purpose of the processing in SIPS is to determine locations in either UTM or latitude/longitude coordinates. The database defines a scheme used to specify relative positions and orientations. The scheme involves using a right-handed coordinate system. Information about the scheme, including the datum, axis definition and units used, is actually stored in the database. Generally, distance units are in meters, time units are in seconds and angles of rotation are in degrees.

Frame of reference

The frame of reference of a rectangular pixel is defined by the edges as a right-handed coordinate system with the origin given by the lower left corner.

Bathymetry and still photographs

While tables for bathymetry and still photographs are not included in the data definition, it is anticipated that these (and the table of known objects) will include references to existing tables. A modification of the existing design should not be required.

Quad-trees

Geocoded images are stored in the database as a quad-tree. For a description of the quad-tree structures, see [[4]].

3.2 Interface between IDL and database

The majority of the code in SIPS is written in IDL. IDL creates structures that mirror tables in the database, and communicates with tables using these structures. Data retrieved from the database is stored in these IDL structures. When new data is to be inserted into the database, the data is first stored in the IDL structures, and those structures are then used to populate the appropriate table in the database.

The actual communication with the database is achieved using an external C program called **idl_illustra_socket** that passes SQL commands from its standard input stream to Informix and writes out data and information returned by Informix to its standard output stream. The IDL code communicates with **idl_illustra_socket** using a set of calls to a library called **DBCon**, which spawns the **idl_illustra_socket** program and uses pipes to communicate with it. The **DBCon** library is necessary because the SPAWN command of the Windows version of IDL does not support interprocess communication.

SIPS also uses another library called **large_obj** to handle creation and reading of large objects. This is independent of the **idl_illustra_socket** program, and in fact establishes a separate connection to the database.

See [[4]], [[5]], [[6]] and [[7]] for more information about how IDL communicates with the database.

4 User Interface Design

While Section 2 gave a basic introduction to using SIPS, this section attempts to give more detailed information and provide references to documentation about the various aspects of the current system.

4.1 Input module

The **Input** button on the main window of SIPS is used to input information about source sonar image into the database. Note that the sonar and mux files must exist before using this module. What is actually stored in the database from the sonar file is header information, *not* the image itself. The image is read in as needed.

4.2 Normalization module

The **Normalize** button is used to perform a normalization of the sonar image. The normalization algorithm used can apply gain and offset functions. It also computes the source sonar altitude and compresses the image as part of the process (the user must specify a compression algorithm). Each transducer or channel produces a separate normalized image. Currently, the underlying code only handles two (2) channels of data, and both channels must be normalized at the same time. The module also allows the user to perform the normalization on only a subset of the original image.

4.3 Demultiplexing module

The **Demultiplex** button is used to demultiplex the navigation data contained in the relevant mux file. The actual demultiplexing is done using Ada code. The user must specify, in addition to the survey pass, a *selection group.*

A selection group specifies which variables should be demultiplexed, and is represented by a record in the **demux_sel_group** table of the database. By default there is a single group in the table. Currently, there is no way to enter new selection groups from the user interface.

Before the demultiplexed data is stored in the database, the user is given an opportunity to plot the data.

4.4 Navigation estimation module

The **Navigation Estimation** button is used to process the demultiplexed data. The demultiplexed data consists of a set of variables each represented as a time series. These variables need to first be synchronized, and then any filters necessary can be applied. The synchronization is performed using the **Process Time Series** button of the Navigation Estimation window. Algorithms are then applied one at a time to the processed time series to apply filters and produce other data.

At any point, any of the variables and/or processed time series can be plotted and even edited manually.

4.5 Towfish track estimation module

The **Towfish Track Estimation** button is used to generate a track estimate for a platform and to produce the six degrees of freedom needed by the geocoding process. Currently, the only track estimation algorithm is the *dead reckoning* algorithm. Note that, at present, this module also produces a georeferenced image.

4.6 Geocoding module

The **Geocode** button is used to produce geocoded images. The user selects one or more georeferenced images to process. Then, for each image, the outline of the track is plotted and the user specifies a rectangular area to be geocoded. The image can then be optionally stored in the database.

4.7 Image display and editing module

The **Image Processing** button is used to display, edit and process images stored in the database. The user can view and print images, choose new colour maps, view image profiles, produce image histograms, and re-sample images. Note that the results of image processing cannot be stored in the database.

4.8 Graphics module

The **Plotting** button is used to plot one-dimensional functions. Currently, the user can only plot data contained in an external file. Time series in the database can be dumped to an external file using the **Dump Plot** button. These external files are simple ASCII files.

See [[4]], [[6]], [[7]] and [[8]] for more information.

5 Changes from SIPS II

SIPS II was implemented on a **DEC Alpha** running **OSF/1** (Unix) with **ENVI** 2.0 and **IDL** 4.1, and used **Illustra** as its DBMS. It contained code written in IDL, C, Fortran and Ada as well as various Unix scripts. The compilers used were standard DEC-supplied compilers.

SIPS III runs under **Windows NT** with **ENVI** 3.1 and **IDL** 5.2, and uses **Informix** as its DBMS. It contains code written in IDL, C and Ada, as well as various NT batch files. With a few modifications, SIPS III should also work under **Solaris**. The compilers used to produce SIPS III were **Microsoft Visual C/C++** and the **ObjectAda** compiler from **Aonix**. Note that the **f2c** program was used to translate the Fortran code in SIPS II to C. Code contained in shared object libraries in SIPS II was put in *Dynamic Link Libraries* (DLLs) in SIPS III. Where possible, compiler directives were used to provide conditional compilation blocks and hence increase portability of the code (for example, by using `#ifdef` C preprocessor directives). All C and Ada code was modified as necessary to compile and run under both Windows NT and Unix.

5.1 Overview of changes

The look and feel of SIPS III is almost identical to that of SIPS II. While the IDL code was not changed significantly, some changes were made to reflect both the change in platform and the change in version of IDL and ENVI.

The **SPAWN** facility works very differently with IDL for Windows. Consequently, the **DBCon** library was created to perform inter-process communication with spawned processes when necessary. Also, the **CALL_EXTERNAL** feature of IDL works differently under Windows, so the IDL code was modified to work differently depending on the platform. Finally, the interface to ENVI has changed since version 2.0, and hence some of the code had to be modified accordingly.

For more information about the changes from SIPS II to SIPS III, see [[1]] and [[2]]. The files containing the code are put under a source code revision system called **CVS**. CVS can be used to view a log of changes for each file in the system if a more detailed analysis of the changes is required.

5.2 Changes to the database

While the basic structure of the database design is the same in both SIPS II and SIPS III, the actual database is completely different because of the change from Illustra to Informix. In particular, the following changes were made to the database:

1. New indices were added to various tables to make queries more efficient.
2. The **not null** clause from the column **axis_definition** in the table **platforms** was removed because of an Informix bug. Note that this clause was never strictly necessary.

3. The type of the **specification** column in the **image_formats** table was changed from **varchar(128)** to **lvarchar**. This is because the value of this column is often a concatenation of 2 strings representing large objects. While under Illustra 128 characters were sufficient to hold such string representations, with Informix these strings often exceed even the 256 character limitation of the **varchar** type.
4. A new table called **quadtree_bufs** was added to store the buffer large object associated with each quadtree. In Informix, unless a large object is inserted into a table (i.e., its *reference count* is greater than 0) it is dropped once the connection that created the large object is closed. So now each buffer large object is inserted into this table. This table has no other significance.
5. A new table called **table_dependencies** was added to help with the auto-sequencing feature. Instead of using triggers, in SIPS III each table is explicitly checked by the program for the lowest available id for a unique row identifier using the MAX() function provided by Informix before any INSERT command is issued. However, because of inheritance properties, the MAX() function must be applied to the top parent table (i.e., the highest level super-table). This table maps each sub-table with its ultimate super-table or parent table. Any time new sub-tables are added, this table should be updated.
6. As mentioned above, the **Poly** type has been implemented as a list of non-null double precision real values (**dreal_t**).
7. Informix does not handle "union views". Therefore, for those views that use the union operator, two separate views have been created, and any time a SELECT operation is issued on the view, the code now constructs the "union view" by applying the union operator to the select statement itself. For an example of this, see the **current_time_series_demuxes** view.
8. The IDL code that parses the textual representation of lists and arrays returned by the database had to be modified. Illustra wraps a list of values or array elements with brackets ([and]), but Informix uses **LIST** { and }. In addition, when inserting values into list and array columns, Illustra did not require any special processing, whereas Informix requires the data to be explicitly cast to the correct data type. New code was therefore added to handle this requirement of Informix.
9. When inserting new rows, SIPS II relied on the Illustra *OID* feature to determine the id of the row just inserted. Since Informix does not have an equivalent feature, extra code had to be written in SIPS III to determine the id of a newly-inserted row.

5.3 Changes to the user interface

As mentioned above, the look and feel of SIPS has not changed, but a few modifications were made in SIPS III. The following general changes were made to the user interface:

1. A facility was added to output time series to external files that could then be plotted.
2. Facilities and buttons in SIPS II that were used purely for debugging purposes were removed from the user interface (although the underlying code was left intact) in SIPS III.

3. Certain code in the system (most notably in the demultiplexing code) used single precision floating point variables to receive double precision values. This was changed where necessary to avoid loss of precision.
4. The **sips_displayfile** IDL procedure was modified so that files larger than 32K can optionally be displayed using an external viewer (such as the **Notepad** application under Windows NT). This was done because it was found that there are log files that could potentially become quite large and hence take an extremely long time to load into a window for display.

In addition, the following sections outline more specific changes made to the user interface.

5.3.1 Colour displays

IDL 5.2 handles displays differently, and hence changes were made so that the system works properly with 24-bit true colour displays under IDL 5.2 for Windows. In addition, any code that changes the colour table was modified to update all windows with images, since this is not done automatically by IDL for Windows on 24-bit displays.

A **Refresh** button was added to the image display window to allow the user to manually refresh the image if necessary.

5.3.2 Running environment

The main program was also modified to read in the values of the following environment variables to set up a proper running environment: **SIPS_RUNPATH, SIPS_NODES, SIPS_VOLUMES, SIPS_BASENAME, SIPS_ENVI_SAVEFILE,** and **SIPS_ENVIM_SAVEFILE**. This feature allows a user to set up these environment variables external to the program itself, and hence avoids having to modify the code when the system is installed on a new machine.

In SIPS III for Windows NT, the **sipsenv.bat** file sets up these environment variables and should be consulted for a working example.

5.4 New files

The following files were added to the system (paths are relative to the top level sips directory). The description and purpose of each file can be viewed using the **cvs log** command.

```
bin/win32/sips.bat
bin/win32/sipsbackup.in
bin/win32/sipsenv.bat
bin/win32/sipsrestore.bat
bin/win32/sipsrestore.in
db/loaddb.bat
db/redodb.sql
db/Sips/createtrig.sql
db/Sips/dropcasts.sql
db/Sips/dropfuncs.sql
obj/makeall.bat
obj/dbcon_proj/dbcon_proj.dep
obj/dbcon_proj/dbcon_proj.dsp
obj/dbcon_proj/dbcon_proj.dsw
obj/dbcon_proj/dbcon_proj.mak
obj/dbcon_proj/dbcontest.c
obj/dbcon_proj/dbconsp/dbconsp.dsp
obj/dbcon_proj/dbcontest/dbcontest.dsp
obj/dbcon_proj/dbcontest/dbcontest.mak
obj/demux_proj/demux_proj.dep
obj/demux_proj/demux_proj.dsp
obj/demux_proj/demux_proj.dsw
obj/demux_proj/demux_proj.mak
obj/geocode_proj/geocode_proj.dep
obj/geocode_proj/geocode_proj.mak
obj/illustra_proj/illustra_proj.dsp
obj/largeobj_proj/largeobj_proj.dep
obj/largeobj_proj/largeobj_proj.dsw
obj/largeobj_proj/largeobj_proj.mak
obj/largeobj_proj/testlo/testlo.dep
obj/largeobj_proj/testlo/testlo.dsp
obj/largeobj_proj/testlo/testlo.mak
obj/lsspline_proj/lsspline_proj.dep
obj/lsspline_proj/lsspline_proj.dsp
obj/lsspline_proj/lsspline_proj.mak
obj/navest_proj/navest_proj.dep
obj/navest_proj/navest_proj.dsp
obj/navest_proj/navest_proj.mak
obj/navest_proj/navest_adalib/navest_adalib.pr
j
obj/norm_proj/norm_proj.dep
obj/norm_proj/norm_proj.dsp
obj/norm_proj/norm_proj.dsw
obj/norm_proj/norm_proj.mak
obj/polygon_proj/polygon_proj.dep
obj/polygon_proj/polygon_proj.mak
obj/quadtree_proj/quadtree_proj.dep
obj/quadtree_proj/quadtree_proj.dsp
obj/quadtree_proj/quadtree_proj.mak
obj/sonar_proj/sonar_proj.dep
obj/sonar_proj/sonar_proj.dsp
obj/sonar_proj/sonar_proj.mak
obj/utm_proj/utm_proj.dep
obj/utm_proj/utm_proj.dsp
obj/utm_proj/utm_proj.mak
src/dbcon/dbcon.c
src/demux/c_types.adb
src/demux/c_types.ads
src/demux/conv.adb
src/demux/conv.ads
src/demux/count_delimit.adb
src/demux/count_delimit.ads
src/demux/demux_adalib.ada
src/demux/field_desc.adb
src/demux/field_desc.ads
src/demux/get_buffer.adb
src/demux/get_buffer.ads
src/demux/namelist.adb
src/demux/namelist.ads
src/demux/pc_vax.ads
src/demux/sips_demux.adb
src/demux/sips_demux.ads
src/demux/string_2_enum.adb
src/demux/upcase.adb
src/envi/display.men
src/envi/e_locate.pro
src/envi/envi.cfg
src/envi/envi.men
src/geocode/rotate_vector.pro
src/illustra/gen_parse_opt.c
src/illustra/genparseopt.h
src/lsspline/eo2bae.c
src/navest/c_types.adb
src/navest/c_types.ads
src/navest/ddt_navest.adb
src/navest/ddt_navest.ads
src/navest/format_io.adb
src/navest/format_io.ads
src/navest/interpol.adb
src/navest/interpol.ads
src/navest/navest_adalib.ada
src/navest/pc_vax.ads
src/navest/sips_navest.adb
src/navest/sips_navest.ads
src/navest/smooth.adb
src/navest/smooth.ads
src/navest/unwrap.adb
src/navest/unwrap.ads
src/norm/normalize.c
src/norm/normio.c
src/norm/normit.c
src/norm/normlib.c
src/norm/translate.bat
src/sips/db/db_error.pro
src/sips/db/db_getid.pro
src/sips/db/db_getmaxid.pro
src/sips/db/db_read_string.pro
src/sips/db/db_read_value.pro
src/sips/graph_module/dump_plot.pro
src/sips/graph_module/get_series.pro
src/sips/graph_module/get_series_names.pro
src/sips/utilities/float2str.pro
src/sips/utilities/call_external/README
src/sips/utilities/graphics/color/sips_xpalette.pr
o
src/utm/ll2utm.c
```

src/utm/root.c

src/utm/utm2ll.c

6 Future Work

While the current implementation of SIPS III has all the features of SIPS II, there are still a number of features in the original SIPS that do not yet exist in SIPS III or SIPS II. Also, there are a few outstanding technical issues that should be addressed, and there are some features that can be added above and beyond those that exist in the original SIPS to make the current system even more useful. The following section describes additions to SIPS III that should be implemented.

6.1 Informix issues

A number of bugs and limitations of Informix were encountered. The bugs have been logged, and the next version of Informix is supposed to overcome the limitations identified. These issues are:

1. Informix puts a limit of 18 characters on identifiers. This restriction did not exist in Illustra. To work around this, the idl_illustra_socket program currently catches long identifiers and translates them according to an internal static table. The new version of Informix is supposed to accept long identifiers.
2. There are two problems encountered with Informix and Windows NT, to do with the **datetime** database type and using **libmi** calls to return non-binary data. A temporary workaround exists now, but when Informix finds a solution, the code will need to be changed accordingly.

Workarounds have been developed for each issue, but eventually, once the bugs have been fixed and the new version released, the new version of Informix should be installed, the bug fixes applied and the workarounds removed.

In addition, there is sometimes a severe performance decrease on certain complex queries. This should be addressed to avoid long waits while processed data is retrieved from the database.

6.2 Solaris port

While a DELL machine has been installed with Solaris, a number of issues prevented a full implementation of SIPS under Solaris, including most importantly the following:

1. Solaris 7 does not yet have drivers for the SCSI adapters or the SCSI hard drives that came with the machine. In addition, there are no drivers that are compatible with those adapters and drives. This made it initially impossible to install Solaris on the machine.
2. An IDE hard drive owned by BCS was put in the DELL machine and used to install Solaris 7, since drivers existed for the IDE bus and that particular drive. However, it was discovered that Solaris 7 also does not yet have drivers for the graphics adapter. This meant that a generic 16 colour driver had to be used, which makes the system functional but not really usable. So, while Solaris 7 has been successfully installed, no

further work was done on developing a Solaris port of SIPS because the operating system does not yet support the latest hardware that shipped with the computer purchased.

In order to carry out a port of Solaris, the following items are required:

1. Drivers for all the hardware on the computer or another computer whose hardware is fully supported by the current version of Solaris.
2. A C compiler (GNU C could be used).
3. An Ada compiler (GNAT could possibly be used).
4. The latest version of Informix for Solaris 7.
5. The latest version of ENVI for Solaris 7.

6.3 Geocoding Module

The geocoding process should be modified to utilize quad-tree indexing during, rather than after, geocoding.

At present the geocoding process assumes a rectangular area, processes the raw sonar data and populates the rectangle. It then creates the index structure and stores the geocoded data. This is impractical for large data sets.

Instead, the index structure should be set up ahead of time. This is possible since the area to be geocoded is known. The georeferenced data has a polygon associated with it. The geocoding algorithm would then allocate areas in memory as they are needed, each area holding one lowest-level image "chip" (approximately 10 KB). The total number of "chips" available at any one time in memory would depend on the available memory. Each area would have a time count. If a new area is needed but memory is exhausted, the oldest area would be posted to the database, and the new area brought in, either from the database if it already exists, or as an empty area. Thus when the geocoding process is complete, the data would already reside in the database. This should be a fairly efficient process if enough memory is available, since it is unlikely that an "old" chip would be needed again once it has been posted to the database.

6.4 Quad-trees

An image data type should be developed that utilizes a quad-tree (R-tree indexing) and has at least the following attributes:

1. Given a receiving buffer, it would retrieve all or a portion of an image such that only data needed to fill the buffer would be used. For example, if the buffer is 1K X 1K, but the image to be retrieved is 100K X 100K, the "get image" function would determine the most efficient way to recover the needed data, interpolate and/or sub-sample if necessary.
2. It would allow for incremental updates of the image. This function could be used by the geocoding process (see above).

6.5 Data import

A routine should be written to "ingest" or import MCDV data. The user interface should be modified to give access to this routine, if possible.

6.6 Object data types

A facility should be added to allow the adding of objects to the database. Objects would include rocks, mines, logs, wrecks, etc. Also a "virtual" object data type should be added. The "virtual" object would be a means of linking objects in different images as a single object. The "virtual" object would have an average position, size, etc. which would be taken from all the instances of the real objects. Depending on the object type there would possibly be a provision to add ancillary data, such as mine type, photos and other information to be determined.

The data import code could serve as a starting point for building a facility for adding objects and their attributes.

6.7 Object identification

A graphical tool should be developed to allow object identification in an image. This capability would be available for either georeferenced or geocoded images. An image would be displayed and any previously identified objects in the image would be highlighted.

The user could then place the cursor on an object. If the object had already been identified, its attributes would be displayed; if unidentified, the user would be able to add the object to the database by indicating its location in the image. The user would also be able to specify the outline of an object to indicate size. The user would also be able to identify shadow length which would be used to automatically calculate object height.

The object would be added to the database with the user supplying an initial object type such as "unknown", "rock", "log", etc. The default would be to classify an object as "unknown".

6.8 Object correlation

A graphical tool should be developed to allow a user to identify objects in two overlapping images simultaneously. The user would select two geocoded images in which there is overlap. The overlapping areas would be shown simultaneously, side by side. When the cursor is placed in one image, a "ghost" cursor would appear in the second image at the same geographic location. All the identified objects in the images would be highlighted such that it would be possible to ascertain whether an object is already in the database. The user could position the cursor on an object, indicate the object in one image (for example by pressing a mouse button), then immediately place the "ghost" cursor on the same object in the second image and indicate that the two objects in fact are the same object (for example by pressing another mouse button). The linkage between objects would be handled by creating a "virtual" object (see Section 6.6). If the object attributes, size, etc. have not been defined as yet, the user would be able to define them at this point.

6.9 Blinking

A "blinking" feature should be added whereby up to four overlapping geocoded images would be displayed alternately in the same window. The blink rate would be adjustable as follows: "static" (i.e., the user would alternate images by pressing a mouse button), 0.5, 1, 1.5, 2, 3, 5, 7.5, 10, 20 and 30 Hz. While the images are "blinking" it would be possible to translate one image relative to the others. This feature exists in the MERIDIAN SIPS (i.e., the original SIPS).

6.10 Track warping

A track warping capability should be developed that utilizes the common occurrence of objects in two or more geocoded images to refine the towfish tracks of the images. Common objects would be identified by the object correlation feature (see Section 6.8). The track warping algorithm has already been developed so what would be required would be to develop an efficient, user-friendly interface.

6.11 Image warping

An image warping facility should be developed. Using the database of objects, one image would be warped such that it would register with a second.

In addition, a "multi-image warp" could be developed such that all overlapping images selected that have common "virtual" objects would be warped to register with each other. The warping would use a minimization approach along the same lines as the track warping feature (see Section 6.10).

6.12 Mosaicking

A mosaic capability should be developed using either a cookie cutter or averaging technique. The MERIDIAN MOSAIC function could be used as a guide. Image registration would not be necessary within this module as it would be handled by the various warping tools. Note that ENVI has a mosaic feature that may be used to make the addition of this feature simple.

6.13 CAD/CAC

A CAD/CAC capability should be added that operates on the whole range of image types: raw sonar, georeferenced and geocoded. Part of this capability would be to check the database to see if objects have already been identified at the same location in other images. This tool should be flexible so that algorithm development would be possible. That is, the actual CAD/CAC algorithms could be in the database and the tool would simply be a framework to allow the user to apply them. This may require an in-depth analysis of algorithm types.

6.14 Output Module

An output module should be developed that would provide a means of printing and otherwise outputting images in various formats (such as PostScript). The images should be annotated appropriately. For example, if the image is geocoded, the longitude/latitude or UTM coordinates should be included. IDL already has the framework to do this, and hence the addition of this feature should not require significant effort.

6.15 Database Administration

An administration tool with a user interface that is easy to use should be developed to facilitate database management. This would minimize the need to use SQL directly through tools such as **dbaccess** and **SQLEditor**.

In addition, currently the database allows the user to duplicate data. For example, the user can create two entirely different normalized images and give them the same name. The program should detect this and warn the user, but still allow the user to continue if desired.

The program should also warn the user if the same operation is performed twice. For example, if the user applies a particular algorithm (such as the dead reckoning algorithm) to a set of variables, and then tries to apply the same algorithm to the same set of variables, the program should warn the user.

6.16 User Interface Improvements

Several improvements can be made to the user interface to make the system easier to use. These features are *not* essential to the proper functioning of SIPS III, nor would they increase the usability of the system. However, they have been identified as features that could potentially save the user time and provide a more user-friendly interface to the processing capabilities of the system.

6.16.1 Tabbing through controls

In a window with multiple controls, the user should be able to use the <TAB> key to cycle through each control as with most Windows-based applications.

6.16.2 Main window

Often after performing an operation in a window in SIPS, the main SIPS window is brought to the forefront (i.e., in front of all other windows). This is annoying and should be changed so that any window is only brought to the front if it is clicked on explicitly by the user.

In addition, a feature should be added that allows the user to monitor the resources being used by SIPS. This may involve showing a window that displays such things as the memory being used, the disk space used by files and the amount of space used in the database.

Finally, it would be helpful to have some way of distinguishing different kinds of windows (other than window names). When SIPS III is first started, an IDL window is displayed along with the main SIPS window. As each operation is performed, other windows are displayed. Also, if ENVI is invoked (e.g., via the image profiling function), one or more ENVI windows are displayed. If each window's icon or title bar colour or other distinguishing feature was set according to the "type" of window (e.g., IDL, SIPS or ENVI), this would make it easier for the user to keep track of the windows currently displayed.

6.16.3 Long processes

There are certain functions (such as geocoding or normalizing an image) that can potentially take a long time to complete. The following features would be useful additions in such cases:

1. When possible, the user should be able to perform the processing in the background.
2. There should be an option to bring up a dialog box after the operation is complete to notify the user.
3. When possible, a progress bar should be shown to let the user know how much of the process has been completed.
4. In all such cases, the cursor should become an "hourglass" during the operation, and return to normal after the completion of the operation. Currently this happens in most cases, but in some cases (most notably when a large image is being loaded into memory for display on the screen) it does not happen.

6.16.4 Selection lists

When the user is presented with a list of items from which to choose, double-clicking on the item should be equivalent to performing some default operation on that item. For example, when choosing from a list of algorithms to apply to a time series, double-clicking on the algorithm should be the same as clicking on the algorithm once and then pressing the OK button.

Also, when multiple selections can be made from a list, the user should be able to use the <SHIFT> and <CTRL> keys to make multiple selections simultaneously as in most standard Windows applications.

6.16.5 Drag'n'Drop

There should be a facility for dragging and dropping files and other objects. For example, dragging a sonar file onto the icon for the SIPS program could cause SIPS to start and present the user with the Input module window; or dragging a time series name onto a plot window could plot that time series.

6.16.6 Large windows

In some cases, the data requested (e.g., an image or all the records in a particular table) is too large to display in a window that fits on the screen. In such cases, if possible, the window should have a scrollbar.

6.16.7 Plotting

The plotting facility should be extended to allow plotting of time series directly from the database rather than just from external files.

7 Appendix

The following sections contain supplementary documentation describing the organization of files, installation and other project notes, schedule of tasks completed and database identifier changes for the SIPS III project.

7.1 Directory Contents

The following gives a brief description of each of the directories in SIPS III (contents of **`dirlist.txt`**).

```
This file briefly describes the various directories in SIPS III.

bin          - executables that the SIPS system uses (not used by the actual
               SIPS system, but are assorted utility programs for sonar files,
               magic files, etc.)
 bin/win32          - Win32 (NT) binaries and batch files

data         - data we received from Dr. Roland Poeckert on tape and ftp
             --various data files (NOT UNDER SOURCE CODE REVISION: comments
               from old SIPS II system written by Brian Corrie)
 data/demux         - directory of files which contain the demuxed time series
                      which correspond to run113 mux data.  This data is in
                      ASCII format. It was delivered by Dr. Roland Poeckert Feb
                      20, 1996 late in the afternoon.
                    - demux related data from Dr. Roland Poeckert
 data/edrd          - directory of assorted data from Dr. Roland Poeckert (data
                      received by tape or ftp from EDRD over the course of the
                      SIPS II project)
                    - one of the earlier tapes we got from Dr. Roland Poeckert,
                      containing 112utm
                    - This seems to be a mixture of stuff, and came from a tape
                      from Dr. Roland Poeckert on September 25, 1995.  The tape
                      consisted of some Fortran code and some mux and navigation
                      data.
 data/navest        - directory of files that were obtained from Dr. Roland
                      Poeckert that contain odds and ends for geocoding and nav
                      estimation.
                    - navigation estimation data from Dr. Roland Poeckert
 data/norm          - normalization data directory
                    - data output from Dr. Roland Poeckert's normalization
                      (copies of all the dud tapes we got)
                    - contains sample input and output for the normalization
                      module. It consists of a source sonar file (run113.sonar),
                      three normalized images (one for each of the normalization
                      techniques), three altitude files (again, one for each of
                      the normalization techniques), and a gain file to use for
                      the normalization.  At a later date, an EDRD employee gave
                      Brian Corrie the altitude files and the gain file in magic
                      dump format, which are included in this directory.
                    - getting this data caused many problems, primarily because
                      of an incompatibility with the VMS version of tar that was
                      used to write the tapes.  In this directory, there are 5
                      subdirectories each of which represents an attempt to
                      retrieve the normalization data from tape.  In the
```

	directory tape5, a new version of tar was used and the files are OK. In tape4 the data was written directly to tape and was restored on the UNIX side with the dd command. The files in the top level directory come from the tape5 subdirectory.
data/runs	- directory that contains run data - main data directory that SIPS uses (where the default scope looks for data). - So far just run_113 and an unknown sonar file is here. - Mostly just the input files (source sonar, mux, gain) are in this directory. Other relevant files are in the norm directory, where the output of the normalization can be found.
data/sonar	- directory with a sonar data file - sonar files from one of Dr. Roland Poeckert's tapes - not sure where this came from, it was originally in the /usr/local/data/SIPS directory. Dimensions are 1229x2666 16 bit data, date of acquisition is 1994-09-23.
data/src	- Source code directory - tape that contained all the ADA, Pascal, Fortran code - contains 102 source code files in Ada, Pascal, and Fortran that make up a good portion of the SIPS system source code. Received on or about December 12, 1995
data/timeseries	- directory of processed time series - output from Dr. Roland Poeckert's demux code
db	- database directory. This is where all the SQL files are kept. There are currently (in SIPS II) 6 different directories corresponding to different versions of the DB.
db/Sips	- SIPS III database directory
lib	- directory where all the shared libraries are kept that SIPS loads in at run time with call_external
lib/alpha	- shared object libraries for DEC Alpha platform
lib/win32	- Win32 (NT) DLLs
man	- manuals/documentation (empty for now)
obj	- directory where the Unix object files (.o files) and Microsoft Visual C/C++ project files, object files and intermediate files for the system are kept (each project is in a separate directory)
obj/dbcon_proj	- DBCon library project files
obj/dbcon_proj/dbconsp	- DBCon sub-project files
obj/dbcon_proj/dbcontest	- project files for test program for DBCon library
obj/demux_proj	- project files for demux.dll
obj/demux_proj/demux_adalib	- project files for demux_adalib.dll (DLL containing Ada code used by demux.dll)
obj/demux_proj/demux_clib	- project files for demux_clib.dll (DLL containing C code used by demux_adalib.dll)
obj/geocode_proj	- project files for geocode.dll
obj/illustra_proj	- project files for the idl_illustra_socket program
obj/largeobj_proj	- project files for largeobj.dll
obj/largeobj_proj/testlo	- project files for test program for the largeobj.dll library
obj/lsspline_proj	- project files for lsspline.dll

```
  obj/navest_proj  - project files for navest.dll
    obj/navest_proj/navest_adalib - project files for navest_adalib.dll (DLL
                                    containing Ada code used by navest.dll)
  obj/norm_proj    - project files for norm.dll
    obj/norm_proj/normit          - project files for normit program
  obj/polygon_proj - project files for polygon.dll
  obj/quadtree_proj -   project files for quadtree.dll
  obj/sonar_proj   - project files for sonar.dll
  obj/utm_proj     - project files for utm.dll

src           - all of the source code
  src/ada          - experimental ada directory containg ADA->C experiments
                     etc.
    src/ada/ada_to_c             - ADA->C experiments
    src/ada/ddtdemux             - ADA->C experiments
  src/dbcon        - source code for dbcon.dll
  src/demux        - code for the ADA demux library
  src/envi         - placeholder for ENVI menu/configuration files
  src/geocode      - code for geocoding algorithm
  src/idl          - IDL related utility programs/functions for reading magic
                     files etc.
    src/idl/util                 - more IDL code
  src/illustra     - illustra socket program
  src/largeobj     - Large object IO library
  src/lsspline     - code for lsspline library
  src/magic        - magic utility code
  src/navest       - code for the ADA navigation estimation library
  src/norm         - code for normalization library
  src/polygon      - code for creating the geocoding polygon
  src/quadtree     - code for creating, writing and reading quadtree data
  src/sips         - main directory for the SIPS code. This directory contains
                     almost all of the IDL code
    src/sips/db                    - code for accessing the database
    src/sips/demultiplexing_module - code for demultiplexing
    src/sips/forms                 - code for presenting user with forms
    src/sips/geocode_module        - code for geocoding
    src/sips/graph_module          - code for plotting data
    src/sips/image_module          - code for image display and processing
    src/sips/input_module          - code for inputting survey information
    src/sips/navestimate_module    - code for navigation estimation
    src/sips/normalization_module  - code for normalization
    src/sips/scope_module          - code for specifying scope
    src/sips/table                 - code for accessing database tables
    src/sips/towestimate_module    - code for towfish track estimation
    src/sips/utilities             - various utilities
      src/sips/utilities/call_external     - utilities for using external
                                             programs
      src/sips/utilities/compound_widgets  - customized compound widgets
      src/sips/utilities/graphics          - graphics utilities
        src/sips/utilities/graphics/colour  - customized versions of XPALETTE
                                              and XLOADCT
        src/sips/utilities/graphics/rect    - code for 'rubber banding'
        src/sips/utilities/graphics/rwin    - code for 'rubber banding'
  src/sips/win_dumps              - placeholder for window dumps
src/sonar        - code for sonar image IO routine library (call external)
```

```
src/util        - other assorted utility programs
src/utm         - code for utm to long/lat conversion library
```

7.2 Database identifier changes

While Illustra accepted identifiers of any length for table, column and view names, Informix restricts the length of such identifiers to a maximum of eighteen (18) characters.

The **idl_illustra_socket** program that communicates between the IDL code in SIPS and the Informix database stores a static table of all identifiers longer than 18 characters. Whenever a SQL command is issued, if any identifier in the command matches an entry in that table, it is replaced with the shortened version before the command is passed to Informix. In future versions of Informix, this limitation should be removed. At that time, the **idl_illustra_socket** program should be modified accordingly. Also, the database schema will have to be modified to use the long names again at that point. ***Note that this will mean the database will have to be rebuilt.***

The following table lists the current SIPS/Alpha Illustra identifiers that have been renamed due to the 18-character limit on Informix identifiers.

7.2.1 Identifier Types

The following mnemonics classify each identifier:

Type:	row type or distinct type
Tab:	table
TabC:	table column
TypeC:	row type column
ViewC:	view column

7.2.2 Table of Changes

The following table details the changes made.

Illustra Identifier	**Informix Identifier**	**Identifier Type**
beam_characteristics	bm_chars	TabC
compression_algorithm	compression_alg	TabC
control_point_match	cp_match	TabC
control_point_match_t	cp_match_t	Type
control_point_matchs	cp_matchs	Tab
control_point_set	cp_set	TabC
control_point_set_1	cp_set_1	TabC
control_point_set_2	cp_set_2	TabC
control_point_set_geocode_t	cp_set_geocode_t	Type
control_point_set_geocodes	cp_set_geocodes	Tab
control_point_set_georef_t	cp_set_georef_t	Type
control_point_set_georefs	cp_set_georefs	Tab

control_point_set_t	cp_set_t	Type
control_point_sets	cp_sets	Tab
current_time_series_demuxes	current_ts_demuxes	V
current_time_series_synchs	current_ts_synchs	V
demux_keyword_length_t	demux_kw_length_t	Type
demux_keyword_lengths	demux_kw_lengths	Tab
demux_map_entry_delim_t	demux_me_delim_t	Type
demux_map_entry_delims	demux_me_delims	Tab
demux_map_entry_fixed_t	demux_me_fixed_t	Type
demux_map_entry_fixeds	demux_me_fixeds	Tab
demux_selection_group	demux_sel_group	TabC
demux_selection_group_t	demux_sel_group_t	Type
demux_selection_groups	demux_sel_groups	Tab
deployment_angle_incr	deploy_angl_incr	TabC
deployment_angle_init	deploy_angl_init	TabC
function_adaptive_t	func_adaptive_t	Type
function_adaptives	func_adaptives	Tab
geocoded_image_georef_t	geoc_img_georef_t	Type
geocoded_image_georefs	geoc_img_georefs	Tab
geocoded_image_mosaic	geoc_img_mos	TabC
geocoded_image_mosaic_gi_t	geoc_img_mos_gi_t	Type
geocoded_image_mosaic_gis	geoc_img_mos_gis	Tab
geocoded_image_mosaic_t	geoc_img_mos_t	Type
geocoded_image_mosaics	geoc_img_moss	Tab
geocoded_image_warp_t	geoc_img_warp_t	Type
geocoded_image_warps	geoc_img_warps	Tab
geocoding_algorithm	geoc_algorithm	TabC
georeferenced_image	georef_image	TabC
georeferenced_image_t	georef_image_t	Type
georeferenced_images	georef_images	Tab
location_description	location_desc	TabC
magnetic_variation	magnetic_var	TabC
magnetic_variation_t	magnetic_var_t	Type
magnetic_variations	magnetic_vars	Tab
number_coefficients	num_coefficients	TabC
platform_config_sensor_t	pf_config_sensor_t	Type
platform_config_sensors	pf_config_sensors	Tab
sub_sensor_function_t	ss_function_t	Type
sub_sensor_functions	ss_functions	Tab
sub_sensor_navigational_t	ss_nav_t	Type
sub_sensor_navigationals	ss_navs	Tab
sub_sensor_tran_bitmap_t	ss_tran_bm_t	Type
sub_sensor_tran_bitmaps	ss_tran_bms	Tab
sub_sensor_transducer_t	ss_transducer_t	Type
sub_sensor_transducers	ss_transducers	Tab
sub_sensor_type_name	ss_type_name	ViewC
survey_pass_time_serieses	survey_pass_ts	V
time_series_in_ping	ts_in_ping	ViewC
time_series_in_error	ts_in_error	ViewC
time_series_demux_t	ts_demux_t	Type
time_series_demuxes	ts_demuxes	Tab
time_series_process_t	ts_process_t	Type
time_series_processes	ts_processes	Tab
time_series_pseudo_t	ts_pseudo_t	Type
time_series_pseudos	ts_pseudos	Tab
time_series_synch_t	ts_synch_t	Type
time_series_synchs	ts_synchs	Tab
time_series_track_t	ts_track_t	Type
time_series_tracks	ts_tracks	Tab

transducer_beam_fixed_t	transd_bm_fixed_t	Type
transducer_beam_fixeds	transd_bm_fixeds	Tab
transducer_beam_varying_t	transd_bm_varyg_t	Type
transducer_beam_varyings	transd_bm_varygs	Tab
transition_distance	transition_dist	TypeC

Note: The following views have each been split into separate views:

current_tsd_ids → current_tsd_id_1 and current_tsd_id_2
current_ts_demuxes → current_ts_demux_1 and current_ts_demux_2
current_ts_synchs → current_ts_synchs1 and current_ts_synchs2

8 References

[1] SIPS Project Plan

[2] Sonar Image Processing System II Upgrade Phase 1: Selection of Computing Platform (Dec/98)

[3] Sonar Image Processing System Development Database Implementation (Mar/96)

[4] Sonar Image Processing System Core Implementation II (Mar/96)

[5] Sonar Image Processing System Development - Core Implementation "Interim Report" (Jan/96)

[6] Sonar Image Processing System Data Definition and User Interface Design (May/95)

[7] Sonar Image Processing System Development Project (Mar/95)

[8] Sonar Image Processing System Development Core Implementation User's Manual

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document.. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.) Defence Research Establishment Atlantic, P.O. Box 1012, Dartmouth, N.S. B2Y 3Z7	2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable). UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title).

Sonar Image Processing System II Upgrade: Phase 1: selection of Computing Platform, Phase 2: Porting and implementation of System III

4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)

Author(s) Barrodale Computing Services Ltd.

5. DATE OF PUBLICATION (month and year of publication of document) May 1999	6a. NO .OF PAGES (total containing information Include Annexes, Appendices, etc). 46 (approx.)	6b. NO. OF REFS (total cited in document) 8

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered).

CONTRACTOR REPORT – DREA CR 1999-039

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include address).
Defence Research Establishment Atlantic
PO Box 1012
Dartmouth, NS, Canada B2Y 3Z7

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant). 1DA	9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written). W7707-8-6103/001/HAL
10a ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b OTHER DOCUMENT NOs. (Any other numbers which may be assigned this document either by the originator or by the sponsor.) DREA CR 1999-039

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)
(X) Unlimited distribution
(   ) Defence departments and defence contractors; further distribution only as approved
(   ) Defence departments and Canadian defence contractors; further distribution only as approved
(   ) Government departments and agencies; further distribution only as approved
(   ) Defence departments; further distribution only as approved
(   ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected).

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

Barrodale Computing Services, under contract to DREA/EDRD, had previously implemented sonar image processing algorithms on a DEC/Unix platform utilizing the ILLUSTRA database system and ENVI/IDL software. Since the end of those contracts, there have been changes to the database and image processing packages as well as a significant increase in computing power available in relatively inexpensive PC-platforms and workstations. The goal of this contract was the porting of the previously developed software to new, more powerful computers utilizing the current versions of the database and image processing software. The contract consisted of two phases: (1) the analysis of the best hardware-software configuration for the system (2) the purchase of the suggested hardware/software and the implementation of the sonar image processing system. The first part of the report – Phase I describes the various computer options for the system in terms of cost and performance. The second part of the report – Phase II – describes the implementation of the sonar imaging software on the new system, includes a user's manual, and has suggestions for future work.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title).

Sonar, image processing, database

512061